



@lunserv

Пользователь

15,0

карма

0,0

рейтинг



Профиль

2

Публикации

16

Комментарии

0

Избранное

1

Подписчики

30 сентября 2013 в 23:20

Разработка → Линейное представление октодерева с использованием кода Мортон

из песочницы

Алгоритмы*, Разработка игр*

Октодерево называют древовидную структуру данных, каждый узел которой имеет восемь потомков. Октодерева применяются для пространственной индексации, обнаружения столкновений в трехмерном пространстве, определения скрытых поверхностей, в трассировке лучей и методе конечных элементов.

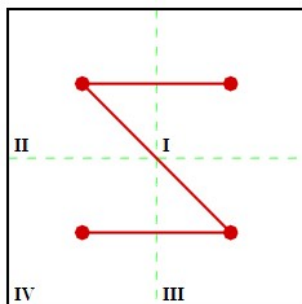
Октодерево может быть представлено в виде иерархической структуры данных или линейно.

В иерархической структуре данных имеется корень дерева, узлы и листья. Каждый узел хранит указатели на его потомков.

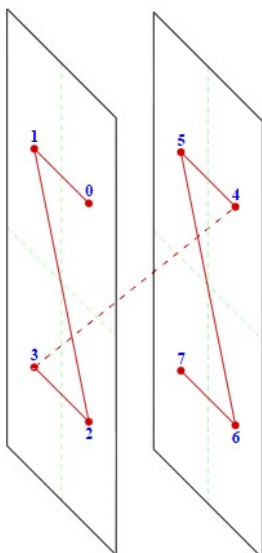
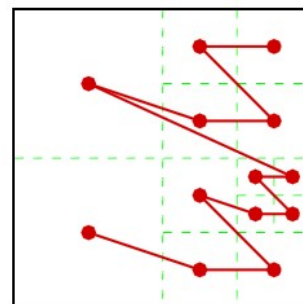
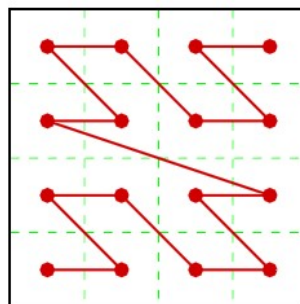
При линейном представлении указатели не используются — применяются различные способы кодирования элементов и хранятся только листья дерева.

Одним из наиболее распространенных и эффективных способов кодирования является применение кривой Лебега (Z-кривой) и применение кривой Гильберта.

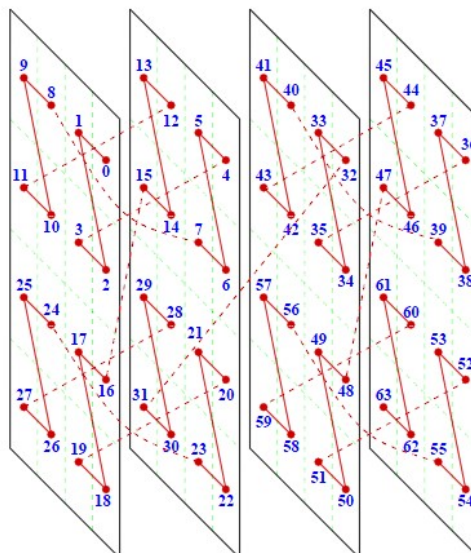
Достоинством кривой Гильберта является ее непрерывность — соседние элементы расположены последовательно. Преимуществом Z-кривой является простота и скорость вычисления, поэтому она чаще применяется на практике.

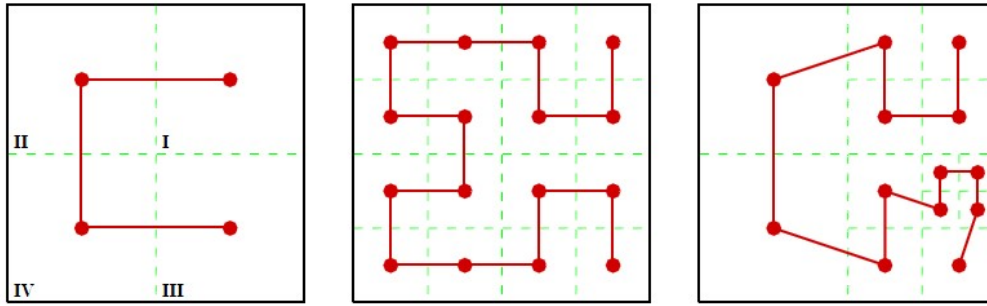


Двумерная кривая Лебега

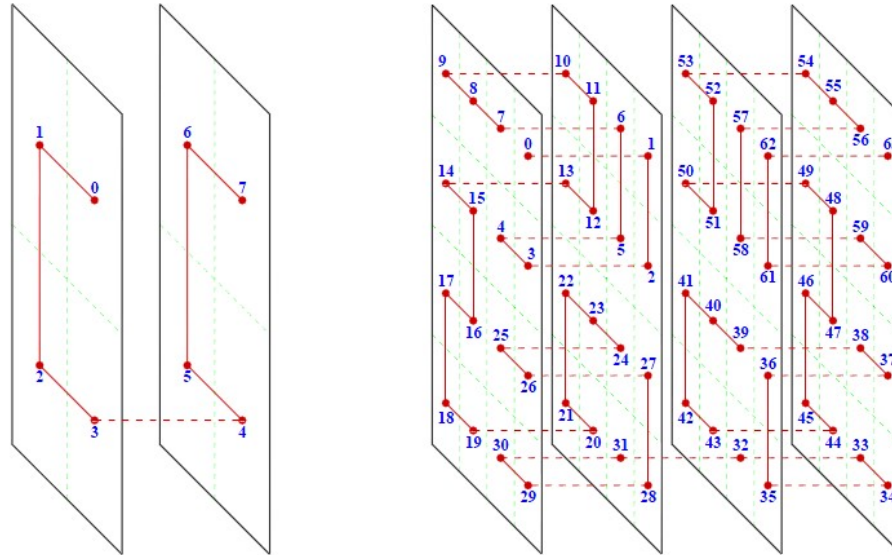


Трехмерная кривая Лебега





Двумерная кривая Гильберта



Трехмерная кривая Гильберта

Для кодирования элементов с использованием Z-кривой используется код Мортон(обычно код вычисляется для узла с минимальными координатами). Код Мортон для Z-кривой вычисляется смещением и смешиванием бит двоичного представления каждой из координат.

	x:	0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
y:	0	000000	000001	000100	000101	010000	010001	010100	010101
	1	000010	000011	000110	000111	010010	010011	010110	010111
	2	001000	001001	001100	001101	011000	011001	011100	011101
	3	001010	001011	001110	001111	011010	011011	011110	011111
	4	100000	100001	100100	100101	110000	110001	110100	110101
	5	100010	100011	100110	100111	110010	110011	110110	110111
	6	101000	101001	101100	101101	111000	111001	111100	111101
	7	101010	101011	101110	101111	111010	111011	111110	111111

Пример вычисления кода Мортон

С учетом свойств Z-кривой, для элементов необходимо хранить только глубину элемента (уровень в октодре) и его порядок (расположение на Z-кривой). В элементы используемого массива для хранения ячеек сетки заносятся значения глубины элемента, а индекс элемента определяет его расположение на Z-кривой.

Для хранения глубины элемента достаточно использовать 1 байт(глубина дерева 256). Для многих задач может оказаться достаточной глубина дерева 16(размер минимальной ячейки будет в $2^{15} = 32768$ раз меньше исходной области). Тогда для хранения ячейки достаточно использовать 4 бита.

Для определения вещественных координат элемента необходимо выполнить следующие шаги:

1. вычисление кода Мортон для элемента

2. декодирование
3. перевод полученного индекса в вещественное значение каждой из координат

Рассмотрим алгоритм на примере кодирования каждой из координат 20-тью битами, то есть результирующий код всех трех координат будет занимать 60 бит.

Зная код и глубину предыдущего элемента, можно вычислить код текущего элемента. Код первого элемента всегда равен 0. Определим смещение для каждого уровня глубины:

```
for ( unsigned char i = 0; i < 21; ++i ) {  
    levelOffset[20 - i] = offset;  
    offset *= 8;  
}
```

Теперь будем определять индекс элемента через глубину и индекс предыдущего элемента:

```
unsigned long getElementIndex( const unsigned long prevIndex, const unsigned char prevLevel ) {  
    return prevIndex + levelOffset[prevLevel];  
}
```

Функция декодирования:

```
void decodeIndexXYZ( const unsigned long index, unsigned long iXYZ[3] ) {  
    iXYZ[0] = decodeIndex( index );  
    iXYZ[1] = decodeIndex( index >> 1 );  
    iXYZ[2] = decodeIndex( index >> 2 );  
}  
  
unsigned long decodeIndex( const unsigned long index ) {  
    unsigned long ind = index & 0x0249249249249249;  
  
    ind = ( ind | ( ind >> 2 ) ) & 0x00C9219243248649;  
    ind = ( ind | ( ind >> 2 ) ) & 0x00386070C0E181C3;  
    ind = ( ind | ( ind >> 4 ) ) & 0x0003E007C00F801F;  
    ind = ( ind | ( ind >> 10 ) ) & 0x000000FFC00003FF;  
    ind = ( ind | ( ind >> 20 ) ) & 0x000000000000FFFF;  
  
    return ind;  
}
```

iXYZ[0], iXYZ[1], iXYZ[2] — определяют порядок кодирования(в данном случае сначала координата x, затем y, затем z).

Константы и количество шагов в функции decodeIndex определяются количеством бит и размерностью пространства(в данном примере константы приведены для трехмерного пространства и 20-ти бит на координату). Существуют различные способы кодирования, примеры на

[Bit Twiddling Hacks](#)

[How to compute a 3D Morton number \(interleave the bits of 3 ints\)](#)


Для получения вещественных значений вершины ячейки с минимальными координатами, полученные индексы умножаются на величину шага. Величина шага — это размер минимальной допустимой ячейки сетки.

Размер ячейки можно определить по ее глубине. Остальные значения определяются прибавлением размера элементы к соответствующим минимальным координатам.

Деление элемента осуществляется путем увеличения его глубины и вставки после него 7 элементов этой же глубины.





Объединение — уменьшение глубины и удалении последующих 7 элементов.


Октодерево является активно изучаемой структурой данных и алгоритмы работы с ним(поиск соседей, интервалов, визуализация и тд) становятся темой докторских диссертаций и научных исследований.

 октодерево, octree, Morton code

↑ +28 ↓

👁 8,7k ⭐ 125





@lunserv

карма 15,0 рейтинг 0,0

ОНЛАЙН КУРС

Защита баз данных Oracle

Александр Шелемин
Разработчик,
Veeam Software

Рустам Ковхаев
Руководитель службы поддержки,
Veeam Software

Евгений Зосимов
Системный инженер,
Veeam Software

СМОТРЕТЬ

Самое читаемое

Сейчас Сутки Неделя Месяц

- +12

Как перенести центр разработки из России в Чехию

15,3k

78

103
- +23

Быстрое клонирование объектов в JavaScript

6,3k

68

31
- +7

Что разработчики интерфейсов могут почерпнуть из японских видеоигр начала девяностых

11,7k

46

4
- +15

Шестое чувство Facebook

12,3k

38

13
- +6

Событийная модель на основе async и await

1,2k

24

0

Комментарии (28)

- Mrrl 1 октября 2013 в 08:38 #

+1 ↑ ↓
- А почему бы декодирование не выполнять за одно умножение с последующим восстановлением порядка битов по таблице:
- ```

unsigned long decodeIndex(const unsigned long index) {
 unsigned long ind = index & 0x0249249249249249;
 ind=(ind*0x10000100001)>>40);
 return DecodeTable[(int) ind&0xFFFFF];
}

```
- Должно получиться не хуже, чем 5 сдвигов и 10 логических операций. Правда, теряется 4 мегабайта памяти :(
- DrSmile 1 октября 2013 в 12:31 # h ↑

0 ↑ ↓
- Операции с некешированной памятью — самые дорогие в современных процессорах. Умножения/сложения/разные битовые операции — наоборот, самые дешевые.
- Mrrl 1 октября 2013 в 14:35 # h ↑

0 ↑ ↓
- Действительно. Пришлось разбить табличку на две — отдельно для старших 10 бит, отдельно для младших. В итоге вычисления по табличкам получились в 2.3 раза быстрее, чем сдвигами (на 64-битной сборке): распаковка 10^9 точек по табличкам — 3.3 сек, сдвигами — 7.7 сек.
- aleks\_raiden 1 октября 2013 в 12:13 #

-1 ↑ ↓
- Lertmind 1 октября 2013 в 12:23 # h ↑


+2 ↑ ↓
- Октодеревья чаще используют в геймдеве.
- aleks\_raiden 1 октября 2013 в 12:26 # h ↑

0 ↑ ↓
- Где, как, для чего, по каким соображениям — этого в статье ничего нет, потому и спрашиваю

 Mrrl 1 октября 2013 в 14:47 #

0 ↑ ↓

Я тоже не очень понимаю, как можно использовать код Мортон для хранения. Как, например, хранить облако точек в виде octree — понятно: разбиваем область на 8 кубиков, в 8 бит записываем маску непустых кубиков, дальше для каждого непустого кубика пишем его представление в том же виде (рекурсивно). Если в кубике осталась только одна точка — записываем младшие биты её координат и прочую информацию (цвет, нормаль...). Код Мортон в этой ситуации полезен только для подготовки дерева — точки в такой структуре идут в порядке возрастания их кодов, так что можно их отсортировать, и дальше всё просто. Но как использовать коды именно для хранения информации?


 lunserv 2 октября 2013 в 00:53 # h ↑

0 ↑ ↓

Если я правильно понял, вы описали линейное представление октодерева, кодируемое из иерархического:

```
 000000
 |
=====
000000 001000 010000 011000 100000 101000 111000
 |
=====
010000 010001 010010 010011 010100 010101 010110 010111
```

то есть для листьев сохраняется весь путь по дереву. Чем глубже дерево, тем больше будет этот путь — по 3 бита на уровень ( $2^3 = 8$  частей). Тогда для дерева глубиной 16 необходимо 45 бит на лист + глубина. И код Мортон здесь не нужен. В описываемом в статье способе листья дерева располагаются вдоль кривой (Лебега, Гильберта и тд) и хранятся только их глубина, то есть все дерево из примера будет храниться как [1]2 2]2 2]2 2]2 2]1 1]1 1]1 1]1] (здесь запись x]y означает десятичную запись числа, где каждый элемент представляет байт — 0000]0000) — используется 4 бита на лист. Далее только по этому значению глубины можно вычислить вещественные координаты каждого листа.

 lunserv 2 октября 2013 в 01:08 # h ↑

0 ↑ ↓

в записи дерева ошибка, правильно так:

```
 000000
 |
=====
000000 001000 010000 011000 100000 101000 110000 111000
 |
=====
010000 010001 010010 010011 010100 010101 010110 010111
```

[2]2 3]3 3]3 3]3 3]2 2]2 2]0] (считаем что 0 — отсутствие элемента, нумерация уровней с 1).

 Mrrl 2 октября 2013 в 08:29 # h ↑


0 ↑ ↓

В моём примере это дерево (15 точек) будет храниться так:

1; 0xff; 0; i0; 0; i1; 1; 0xff; 0; i20; 0; i21; 0; i22; 0; i23; 0; i24; 0; i25; 0; i26; 0; i27; 0; i3; 0; i4; 0; i5; 0; i6; 0; i7

Здесь 0 и 1 — биты, указывающие, идёт дальше разбиение на кубик или точка, 0xff — маска, означающая, что все подкубики непустые, i0, i1, i3..i7 — координаты точек, локализованных на верхнем уровне (при глубине дерева 15 это 42 бита), i20, i21..i27 — координаты точек, локализованных на втором уровне (по 39 бит — старшие 2 бита каждой координаты уже известны). Итого на структуру дерева — 39 бит (против 60 в вашем примере) плюс 606 бит на уточнение координат.

Если какой-то элемент (подкубик) пропущен (не содержит точек), то в маске в соответствующем месте будет стоять 0, и в последующем списке информации про этот подкубик не будет.

 lunserv 2 октября 2013 в 14:59 # h ↑

0 ↑ ↓

Итого на структуру дерева — 39 бит (против 60 в вашем примере) плюс 606 бит на уточнение координат.

хорошо, в предложенном в статье способе 60 бит не хранятся, а вычисляются. Хранятся только 4 бита на лист — все больше ни чего — ни предыдущие уровни, ни каких масок и тд.

Возможно тогда напишу статью с более подробным описанием самого представления.

 Mrrl 2 октября 2013 в 15:00 (комментарий был изменён) # h ↑

0 ↑ ↓

15 чисел (по одному на лист), по 4 бита каждое — итого 60. Разве нет?

 lunserv 2 октября 2013 в 18:37 # h ↑

0 ↑ ↓

да, думал речь идет про 60 бит (3 по 20) для кода.

я не понял как у вас получилось 39 бит и что значит 606 бит на уточнение координат. то есть по вашему для хранения приведенного выше дерева нужно  $39 + 606 = 645$  бит против 60 предложенных в статье? что будет происходить при более глубоком дереве — нужно сохранять весь путь?

 Mrrl 2 октября 2013 в 18:53 # h ↑

0 ↑ ↓

У меня задача — есть множество точек, заданных координатами (в данном примере 15 точек, координаты

15-битные), и надо их упаковать. Когда в остree я дохожу до области, в которой находится одна точка, то для определения её положения пройденного пути недостаточно — надо указать положение точки в области. И это занимает  $3 \cdot (15-d)$  бит на точку ( $d$  — глубина области в дереве). Сохранять путь должна программа обхода — а как же иначе? В самом дереве путь не хранится, он деревом определяется.

 lunserv 2 октября 2013 в 19:12 # h ↑ 0 ↑ ↓

в приведенном способе обход осуществляется сразу по листьям.

 Mrrl 2 октября 2013 в 19:25 # h ↑ 0 ↑ ↓

Но путь всё равно накапливается (пусть суммированием, а не наращиванием), и сохраняется в программе обхода? То, что в вашем подходе нет рекурсии, и за счёт этого время перехода к следующему листу  $O(1)$  — согласен.

 Mrrl 2 октября 2013 в 15:11 # h ↑ 0 ↑ ↓


Кстати, допускает ли эта структура пропущенные листья? Или её назначение — задать разбиение всего пространства (большого куба)?

 lunserv 2 октября 2013 в 18:38 # h ↑ 0 ↑ ↓

да, это полное разбиение пространства, на этом и основан алгоритм.

 Mrrl 2 октября 2013 в 18:59 # h ↑ 0 ↑ ↓

А, ну, тогда есть решение за  $8/7$  бита на лист дерева, вне зависимости от глубины. Если у нас есть область, не являющаяся листом, то разобьём её на 8 подобластей, и составим 8-битную маску, в которой 0 соответствует подобласти-листу, а 1 — подобласти, которая ещё подлежит разбиению. Код области = [маска | коды подобластей]. В вашем примере дерево кодируется 2 байтами — [0x04, 0x00]

 lunserv 2 октября 2013 в 19:14 # h ↑ 0 ↑ ↓

а что будет при разбиении например каждого листа на предыдущем уровне на 8 частей — придется хранить всю эту информацию, по байту на каждое разбиение.

 Mrrl 2 октября 2013 в 19:28 # h ↑ 0 ↑ ↓


Если будет «разбиение листа», то это уже не лист. Если все узлы 2-го уровня разбиты на 9 листьев, то в дереве будет 64 листа, а в коде — 9 байт: [0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00].  $72/64 = 9/8$  бита на лист.

 lunserv 2 октября 2013 в 19:44 # h ↑ 0 ↑ ↓

хорошо, а если разбить 3-тий уровень? в данном подходе затраты на хранение будут расти с каждым уровнем.

 Mrrl 2 октября 2013 в 19:57 # h ↑ 0 ↑ ↓

Будут. Но никогда не превысят  $8/7$ . Если у нас 512 листьев, то код составляет  $1+8+64=73$  байта, т.е.  $73/64$  бита на лист.

 lunserv 2 октября 2013 в 20:14 # h ↑ 0 ↑ ↓

получается у вас для каждой последовательности должна храниться маска всех предыдущих уровней?

можете привести пример, как в вашем способе будет выглядеть запись дерева, где корень разбит на 8, затем самый левый потомок на восемь и у него самый левый на восемь? и как вычислять координаты листьев при таком представлении?

 Mrrl 2 октября 2013 в 20:44 # h ↑ 0 ↑ ↓

Если я не перепутал число уровней, то код будет таким: [0x01, 0x01, 0x00]

Перебор всех листьев выглядит примерно так:

```
typedef unsigned int uint;
typedef unsigned char byte;
void Process(byte *T, int depth) {
 Process(T, 0, 0, 0, 1u << depth);
}

byte *Process(byte *T, uint x, uint y, uint z, uint level) {
 byte mask = *T++;
 level >>= 1;
 for (int k = 0; k < 8; k++) {
 uint x1 = x + level * (k & 1), y1 = y + level * ((k > 1) & 1), z1 = z + level * ((k > 2) & 1);
 if (mask & 1) T = Process(T, x1, y1, z1, level);
 else ProcessList(x1, y1, z1, level);
 }
}
```

```
mask>=1;
}
return T;
}
```

Но я его не проверял, написал первое, что пришло в голову.



**lunserv** 2 октября 2013 в 20:56 (комментарий был изменён)

# h t

0 ↑ ↓

хорошо, у вашего представления свои достоинства, для каких то задач оно подойдет лучше. в статье про [id Tech 6](#) говорится что

является возможным сжатие SVO до уровня 1,15 битов на один воксел

возможно использовалось похожее представление на предложенное вами.



**vladimirovich** 3 октября 2013 в 21:39 (комментарий был изменён)

#

0 ↑ ↓

разные есть способы, кстати.

Вот способ через умножение, правда, 64 битное. 8 битный вход и 16 битный выход

```
unsigned char x; // Interleave bits of (8-bit) x and y, so that all of the
unsigned char y; // bits of x are in the even positions and y in the odd;
unsigned short z; // z gets the resulting 16-bit Morton Number.
```

```
z = ((x * 0x0101010101010101ULL & 0x8040201008040201ULL) *
0x0102040810204081ULL >> 49) & 0x5555 |
((y * 0x0101010101010101ULL & 0x8040201008040201ULL) *
0x0102040810204081ULL >> 48) & 0xAAAA;
```



**lunserv** 3 октября 2013 в 23:51

# h t

0 ↑ ↓

в статье приведена эта ссылка

Только зарегистрированные пользователи могут оставлять комментарии. [Войдите](#), пожалуйста.

## Интересные публикации



- Как мы разработали чат-фреймворк для Android приложения — Chateau 1
- Событийная модель на основе async и await 0
- В Чили так много энергии, что потребители получают ее бесплатно 64
- Повышены штрафы за использование несертифицированных средств связи 91
- Microsoft обвиняют в новом трюке с принудительной установкой Windows 10 51